ARTIFICIAL INTELLIGENCE

Robust flight navigation out of distribution with liquid neural networks

Makram Chahine†, Ramin Hasani†*, Patrick Kao†, Aaron Ray†, Ryan Shubert, Mathias Lechner, Alexander Amini, Daniela Rus

Autonomous robots can learn to perform visual navigation tasks from offline human demonstrations and generalize well to online and unseen scenarios within the same environment they have been trained on. It is challenging for these agents to take a step further and robustly generalize to new environments with drastic scenery changes that they have never encountered. Here, we present a method to create robust flight navigation agents that successfully perform vision-based fly-to-target tasks beyond their training environment under drastic distribution shifts. To this end, we designed an imitation learning framework using liquid neural networks, a brain-inspired class of continuous-time neural models that are causal and adapt to changing conditions. We observed that liquid agents learn to distill the task they are given from visual inputs and drop irrelevant features. Thus, their learned navigation skills transferred to new environments. When compared with several other state-of-the-art deep agents, experiments showed that this level of robustness in decision-making is exclusive to liquid networks, both in their differential equation and closed-form representations.

INTRODUCTION

Intelligence in natural brains differs from today's deep learning systems. The differences are rooted in the ways that these systems learn to make sense of their environment and manipulate it to achieve their goals. Natural learning systems interact with their environments to understand their world. We define understanding as the ability to capture causality and contexts while learning abstract concepts to reason, plan, and control (*1*). This rich representation learning capability allows agents to extrapolate and perform inference and credit assignment even in unseen scenarios and out of distribution (OOD), which is currently beyond the performance achievable with either overparameterized deep learning systems (*2*) or classical statistical learning theory analysis (*3*), because the former overfit to their training data and the latter requires the independent and identically distributed condition on the data distribution.

Studying natural brains effectively narrows the search space of possible algorithms for acquiring intelligent behavior. For instance, neural circuits in brains are much more robust to perturbations and distribution shifts than deep neural networks while also being more flexible in tackling uncertain events compared with deep learning systems. This is because they deploy both unconscious (to facilitate changes in distributions faster) and conscious (to distill causal structure of data and manipulate learned concepts and models) processes (*1*) for decision-making. In contrast, today's deep learning systems are incapable of capturing causality using extracted concepts and features explicitly, despite being able to learn quickly from implicit types of observations.

We find inspiration in the foundational properties of natural learning systems to transform our current deep representation learning frameworks, especially for real-world deployment of artificial learning systems as intelligent agents. In this article, we explore how biologically inspired priors on neural models and network architectures can lead to more flexible, robust, and understandable decision-making for autonomous robots. In particular, we investigate how to construct agents capable of generalizing and achieving zero-shot transfer to new environments for some tasks. To study zero-shot transfer, we focused on end-to-end drone navigation tasks from pixel inputs. For example, consider a scenario where the objective is for a drone agent to navigate to a static target placed in a forest environment. We trained neural network agents on a few runs of data collected by a human pilot in an offline supervised learning setting and observed how the agents transferred under drastic changes in scenery and conditions, such as changing from wild to urban landscapes across multiple seasons. Performing vanilla imitation learning on raw expert demonstrations is sample efficient but generally results in poor closed-loop (active) testing performance due to policy stationarity and compounding errors (*4*).

Specifically, we aimed to develop learning-based solutions to robot flight control that are robust and transferable to novel environments. We studied this problem in the context of the flight hiking task, where a quadrotor robot controller is trained, in one environment, to recognize and navigate toward a target by imitation learning. The resulting policy is then deployed to recognize and move to the target iteratively in different environments. Achieving good out-of-distribution performance requires that trained models learn representations that are causally associated with the task, compositionally resilient, and independent of the environmental context.

There has been extensive research on improving the generalization performance of few-shot, one-shot, and zero-shot imitation learning agents by adopting augmentation strategies (*5*, *6*), human interventions (*7*, *8*), goal conditioning (*9*–*12*), reward conditioning (*13*–*15*), task embedding (*16*), and meta-learning (*17*). These advances help design a better gradient descent–based learning scheme without consideration of the structure of the underlying policy architecture. In contrast, there is evidence that brain-inspired

Massachusetts Institute of Technology (MIT), Cambridge, MA, USA.
*Corresponding author. Email: rhasani@mit.edu
†These authors contributed equally to this work.

neural dynamics improve the robustness of the decision-making process in autonomous agents, leading to better transferability and generalization in new settings under the same training distribution (*18*–*21*). We aimed to leverage brain-inspired pipelines and empirically demonstrate that if the causal structure of a given task is captured by a neural model from expert data, then the model can perform robustly even OOD. In a set of fly-to-target experiments with different time horizons, we show that a certain class of brain-inspired neural models, namely, liquid neural networks, generalizes well to many OOD settings, achieving performance beyond that of state-of-the-art models.

RESULTS

Our objective is to systematically evaluate how robustly neural architectures can learn a given task from high-dimensional, unstructured, and unlabeled data (for example, no object labels, no bounding boxes, and no constraints on the input space) and transfer their capabilities when we change the environment drastically.

Models that can be trained in one environment to perform in a multitude of other environments have a certain degree of robustness, achieving zero-shot generalization. One approach to obtain such control policies is scaling up their size with structural scaling of neural network policies (*22*–*24*). The universal law of robustness (*25*) suggests that we can obtain robust decision-making systems (worst-case robustness) by training a large neural network model of a size of at least $n \times d$, where n is the number of observations and d is the dimensionality of each sample of the collected dataset. This theoretical result agrees with the recent advances in the impressive capabilities of large models (*26*) on zero-shot and few-shot generalization across a wide variety of tasks, such as natural language processing (*27*–*29*), text-to-image generation (*30*–*32*), scene understanding (*33*), and even vision-based control (*23*, *34*).

On resource-constrained embedded systems such as aerial robots, however, it is infeasible to take advantage of the intriguing generalization properties of large models because of their high computational cost of inference. Recently, a class of compact and brain-inspired continuous-time recurrent neural networks has shown great promise in modeling autonomous navigation of ground (*18*, *19*) and simulated drone vehicles end to end in a closed loop with their environments (*21*). These networks are called liquid time-constant (LTC) networks (*35*), or liquid networks. They are formulated by a nonlinear state-space representation that has the ability to account for external and internal interventions during training (*36*). These properties make them a great candidate for achieving OOD generalization.

It is challenging for machine learning agents to extract the cause and effect of a given task from unstructured and high-dimensional observations. In this work, we created a testbed to assess this capability for various learning systems without any guidance. In particular, the agents must learn/understand the task, the scene, and the object from offline data without any guidance on what the scene, object, and task are. We show how to learn the task and scene end to end from data and how to generalize to unseen and heavily shifted environments with liquid neural networks, where modern alternatives fail.

To this end, we performed a series of quadrotor closed-loop control experiments where pretrained neural architectures were tested for their ability to generalize OOD. Our experiments included the following diverse set of tasks:

1) Fly-to-target tasks. Train on offline expert demonstrations of flying toward a target in the forest and test online in environments with drastic scenery changes.

2) Range test. Take a pretrained network from the fly-to-target task and, without additional training (zero-shot), test how far away we can place the agents to fly toward the target.

3) Stress test. Add perturbations in the image space and measure the success rate of agents under added noise.

4) Attention profile of networks. Apply feature saliency computation to assess the task understanding capabilities of networks via their attention maps.

5) Target rotation and occlusion. Rotate and occlude the target object in the environment and measure whether the agents can complete their flight to the target to measure rotational and occlusion invariance.

6) Hiking with adversaries. Take a pretrained network and use their attention profile for hiking from one object to another one in an environment with a drastic change of scenery, environmental perturbations, and adversarial targets.

7) Triangular (multistep) loop between objects. In a fully new environment, hop from one target to others placed in a triangular state and loop from target to target multiple times.

8) Dynamic target tracking. Take a pretrained network and test how well they can track a moving target in two testing environments with different scenery, lighting, and wind conditions.

We first describe the baseline models we compared against each other. We then present our training setup for the fly-to-target task. Last, we go through our comprehensive testing results in zero-shot closed-loop control.

Baseline models

We compared six different recurrent neural network architectures, namely, long short-term memory (LSTM), gated recurrent unit (GRU)–ordinary differential equation (ODE), ODE–recurrent neural network (RNN), temporal convolutional network (TCN), neural circuit policy (NCP), and closed-form continuous-time (CfC), which are briefly described below. LSTM (*37*) is a discrete-time neural network that consists of a memory cell that is controlled by learnable gates that access, store, clear, and retrieve the information contained in the memory. ODE-RNN (*38*) transforms a vanilla RNN network, for example, an autoregressive fully connected network, into a continuous-time model by using the hidden state of the RNN as the initial state of a neural ODE, for example, an ODE whose derivative is parameterized by a neural network. The time continuity enters the model in the form of the simulation time when solving the ODE with a numerical ODE solver. GRU-ODE pairs a discrete-time GRU (*39*) network with a neural ODE, making it a continuous-time version of a GRU. TCNs (*40*) model the temporal structure of an input sequence by performing a one-dimensional convolution over the time dimension. NCP (*18*) is a sparsely wired four-layer architecture built by LTC cells. The architecture is structurally and semantically inspired by the nervous system of the *Caenorhabditis elegans* worm (*19*). The CfC model (*21*) is a sparsely wired four-layer architecture built by numerical approximation of the closed-form solution of LTCs and is faster to compute than ODE-based models.

Fly-to-target task training

The fly-to-target task consists of autonomously identifying a target of interest and performing the flight controls driving the quadrotor toward it using the onboard stabilized camera's sequences of RGB (red, green and blue) images as the sole input (Fig. 1). We initially started the quadrotor about 10 m away from the target and required that the policy guide the drone to within 2 m of the target, with the target centered in the camera frame. The objective was to learn to complete this task entirely from an offline dataset of expert demonstrations. The learned policies were then tested online in a closed loop within the training distribution and in drastically distinct settings. This experimental protocol allows for the principled assessment of performance and generalization capabilities of liquid networks compared with modern deep models (*38*, *41*, *42*).

Closed-loop (online) testing

All seven tasks were evaluated in closed loop with environments by running the neural networks onboard the drone. In the following, we describe our results.

Fly-to-target and zero-shot generalization

In this setting, the quadrotor was placed at a fixed distance from a target of interest seen in the training data. In our evaluation experiments, we used a red camping chair as the target for collision avoidance purposes; because of its relatively large size, it could occupy an important portion of the camera frame without the drone having to get dangerously close. The target was present in the initial field of view, and its position in the initial input frame was randomized but balanced over all runs (each network was tested on the same equal number of right, center, and left starting positions at slightly different altitudes). The initial testing protocol was repeated for a number of testing environments to evaluate the networks' performance in OOD settings but maintained a task structure identical to the one learned from training data. We randomly positioned the quadrotor at a distance of about 10 m from the target, with the latter in the field of view of the onboard camera. We launched the closed-loop policy and observed whether the network could successfully guide the drone to the target. The test was repeated 40 times for each network and in each environment. Success was accounted for when the network was able to both stabilize the drone in a radius of 2 m and maintain the target in the center of the frame for 10 s. Failure cases were identified when the network generated commands that led to an exit of the target from the range of view without the possibility of recovery. We also included cases where the drone failed to reach the target in less than 30 s to account for rare runs where the network generated commands indefinitely, maintaining the drone in a stable position with the target in sight but with no intention of flying toward it.

We tested the policies in four environments we call Training Woods, Alternative Woods, Urban Lawn, and Urban Patio (Fig. 1, B and D). The first corresponded to the target in the woods, respecting roughly the same position and orientation distribution as that of the training set. We thus evaluated the performance of the networks on image sequences from the scenery used during training with different lighting conditions and wind profiles. Alternatively, we moved the chair to a different spot with a different background in a neighboring part of the woods. This experiment evaluated the networks' generalization performance on input data with scenery distribution shifts but maintained proximity to sequences seen during training. To truly evaluate robust task understanding, we subsequently set up the zero-shot domain transfer experiments in a drastically dissimilar environment: a piece of lawn on CSAIL MIT's campus. Although containing green grass and a few trees, the

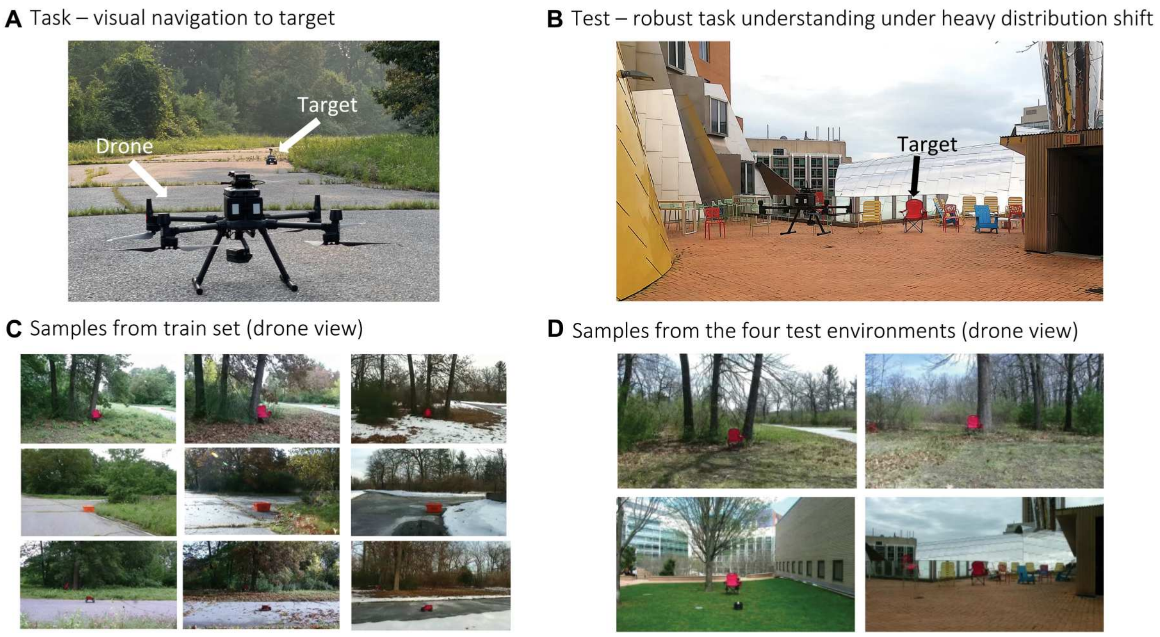
A Task – visual navigation to target
Target
Drone
B Test – robust task understanding under heavy distribution shift
Target
C Samples from train set (drone view)
D Samples from the four test environments (drone view)

Fig. 1. Sample frames of training and test environments. (**A**) Third-person view of the quadrotor and one of the targets in the Training Woods where data were collected. (**B**) Third-person view of the quadrotor during testing on the Urban Patio, with multiple adversary objects dispersed around the target camping chair. (**C**) Training data frame samples from the quadrotor onboard camera containing various targets (camping chair, storage box, and RC car from top to bottom) and taken during different seasons (summer, fall, and winter from left to right). (**D**) Test data frame samples from the quadrotor onboard camera against each of the four test backgrounds: Training Woods (top left), Alternative Woods (top right), Urban Lawn (bottom left), and Urban Patio (bottom right).

landscape was largely different from the woods, as shown in Fig. 1 (A to D). Frames contained buildings, windows, large reflective metallic structures, and the artificial contrast from the geometric shades they induced. Lighting conditions at different times of day and varying wind levels added additional perturbations.

We lastly examined the networks' generalization capabilities in an OOD environment consisting of a brick patio. In this environment, the background, including a number of man-made structures of different shapes, colors, and reflectivity, drastically differed from the training environment. Moreover, we added an extra layer of complexity to this experiment by positioning a number of other chairs in the frame of different colors (including red) and sizes. This ultimate test, including real-world adversaries, required robustness in the face of extremely heavy distribution shifts in addition to proper task understanding to recognize and navigate toward the correct target. Table 1 shows the success rates of the different network architectures across the four environments. Both variants of liquid networks outperformed other models in terms of task completion. Moreover, we observed that, generally, recurrent networks worked better when tested within the same training distribution, such as the Training Woods and Alternative Woods environment. Meanwhile, in OOD scenarios, their performance dropped considerably. However, in such OOD settings, liquid networks stood out, with a sparse liquid CfC network having a success rate of 90% in Urban Lawn and 67.5% in Urban Patio (which contains more natural adversaries).

Range tests

Beyond the extensive testing from the fixed distance of 10 m from the target (which was roughly the starting position in the training data), we also performed a range test to explore how the networks react to an extension of the task unseen in the training data. The experimental scenery was that of the Training Woods setup presented above but with the drone's starting positions growing in distance to the target, thus requiring an increasingly acute task understanding and perception capacity to successfully achieve the objective. We ran 10 experiments for each of 10-, 20-, and 30-m distance from the target. At the time of testing, the target lay in the shade, making it hard to detect against the brighter components in the image, such as tarmac, sky, and trees. The performance of each of the networks is summarized in Table 2.

The results of this experiment show strong evidence that liquid networks have the ability to learn a robust representation of the task they are given and can generalize well to OOD scenarios where other models fail. This observation is aligned with recent works (*21*) that showed that liquid networks are dynamic causal models (DCMs) (*43*) and can learn robust representations for their perception modules to perform robust decision-making. In particular, CfC networks managed to fly the drone autonomously from twice and three times the training distance to their targets from raw visual data with a success rate of 90 and 20%, respectively. This is in contrast to an LSTM network that lost the target in every single attempt at both these distances, leading to a 0% success rate. Only ODE-RNN managed to achieve a single success at 20 m among the nonliquid networks, and none reached the target from 30 m.

Stress tests

In addition to the flight experiments with the networks running online, we performed a series of offline stress tests as another measure of the robustness of different neural architectures to distribution shifts by perturbing the input frames to the network and observing changes in network outputs. Ideally, networks should maintain high performance despite varied environmental conditions. To simulate this domain randomization, we perturbed images from collected sequences, changing brightness, contrast, saturation, and noise to mimic lighting differences, color variations, and sensor noise found in real-world testing. We then measured whether networks substantially changed their control outputs, indicating failure to generalize to new environments, or generated consistent outputs across different perturbations, displaying resilience to distribution shifts.

To quantify the difference in network outputs, we applied a perturbation to the frames of a recorded sequence, fed the original and perturbed frames to the network, and recorded the original and perturbed output velocities generated by the network. We then applied Forward Euler numerical integration to integrate the velocities into position trajectories. We thus measured the final distance between the last point on the two trajectories, recorded this as the perturbed distance, and repeated this process for different perturbation types and amplitudes. Figure 2 shows the results of stress testing for noise, brightness, contrast, and saturation perturbations.

In all four cases, CfCs demonstrated superior robustness compared with their counterparts. They deviated the least from their nominal trajectories. This observation is consistent with our

Table 1. Closed-loop evaluation of the fly-to-target task for the trained policies in four testing environments. Quantitative results are success rates. Higher is better. ($N = 40$)

Algorithm	Environment			
	Training Woods	Alt. Woods	Urban Lawn	Urban Patio
LSTM	82.5%	92.5%	62.5%	27.5%
GRU-ODE	50%	17.5%	32.5%	17.5%
ODE-RNN	62.5%	82.5%	17.5%	25%
TCN	7.5%	0%	0%	0%
NCP (ours)	100%	95%	57.5%	52.5%
CfC (ours)	85%	95%	90%	67.5%

Table 2. Range test evaluation of the trained policies in the Training Woods environment. Quantitative results are success rates. Higher is better. ($N = 10$)

Algorithm	Initial distance (m)		
	10	20	30
LSTM	100%	0%	0%
GRU-ODE	70%	0%	0%
ODE-RNN	80%	10%	0%
TCN	5%	0%	0%
NCP (ours)	100%	50%	10%
CfC (ours)	100%	90%	20%

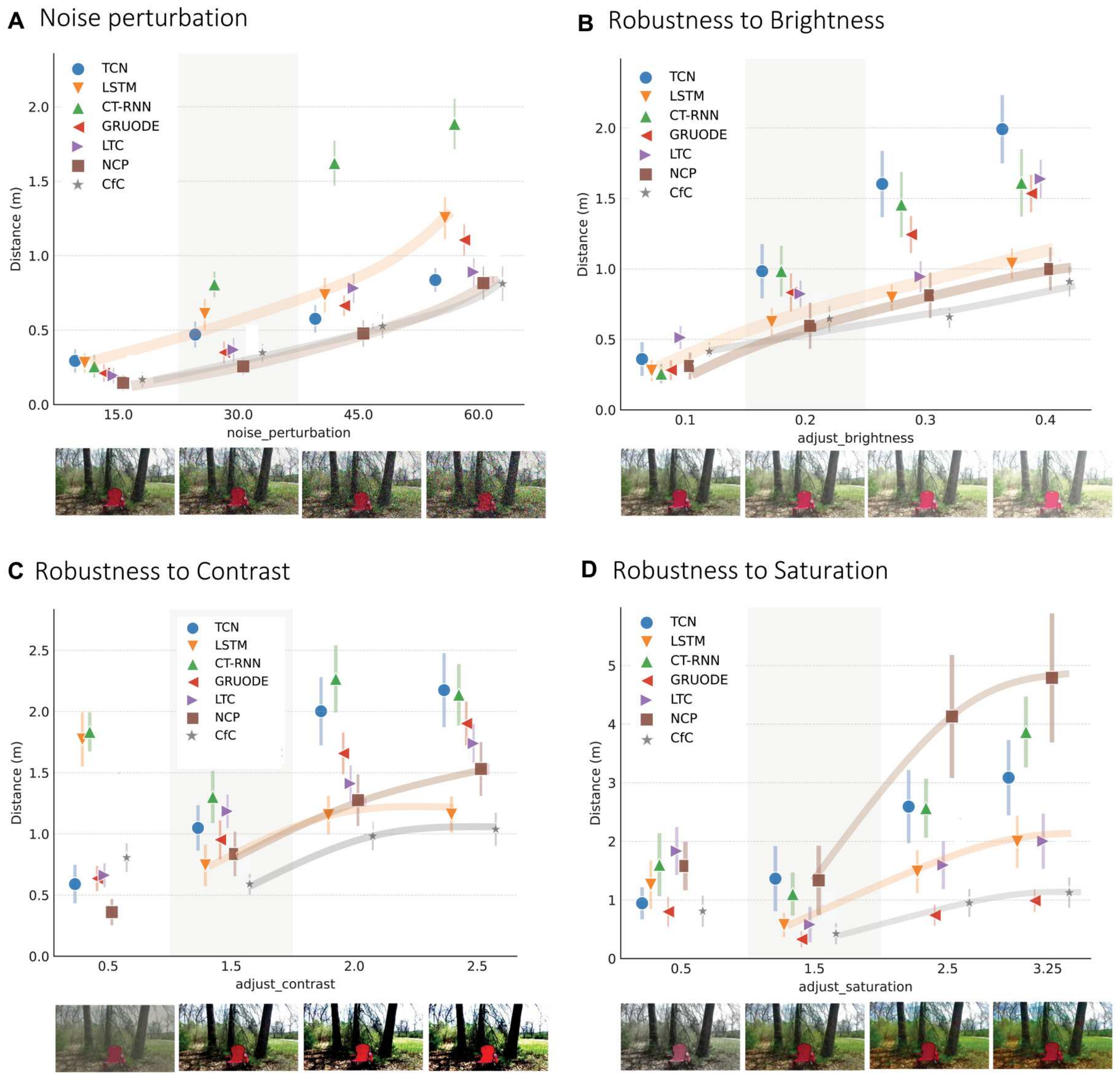

Fig. 2. Noise, brightness, contrast, and saturation stress tests. These experiments run both an original image sequence collected by the drone and a perturbed version through a network, integrate the output velocity commands, and record the final displacement between sequences (lower is better). Bars show the SD across 10 runs. Trendlines for NCP, LSTM, and CFC are shown with shaded brown, orange, and gray lines, respectively ($N = 10$). (**A**) Distances when Gaussian noise with SD *x* is added to image frames. (**B**) Distances when brightness is shifted by *x* times the maximum pixel value. (**C**) Distances when the contrast in the image is multiplied by *x*. Note that the first data point, 0.5, is actually a reduction in contrast and moves all pixel values toward the mean, whereas the other three data points increase contrast and move pixel values away from the mean. (**D**) Distances when saturation (*S* in HSV colorspace) is multiplied by *x*. Note the first data point, 0.5, reduces the saturation of colors in the image, whereas the other three data points increase saturation.

active testing results, where in OOD scenarios, CfCs stood out in task completion success rate. Liquid NCPs also showed great resiliency to noise, brightness, and contrast perturbations because their performance was hindered by changes in input pixels' saturation level. In this offline setting, LSTMs were also among the most robust networks after liquid networks. However, their OOD generalization was poor compared with liquid networks.

Attention profile of networks

To assess the task understanding capabilities of all networks, we computed the attention maps of the networks via the feature saliency computation method called VisualBackProp (*44*) to the CNN backbone that precedes each recurrent network. VisualBackProp associates importance scores to input features during decision-making. The method has shown promise in real-world robotics applications where visual inputs are processed by convolutional filters first. The attention maps corresponding to the convolutional layers of each network in a test case scenario are shown in Fig. 3. We observed that the saliency maps of liquid networks (both NCPs and CfCs) were much more sensitive to the target from the start of the flight compared with those of other methods.

We show more comprehensive saliency maps in different scenarios in figs. S1 to S3. We observed that the quality of the attention

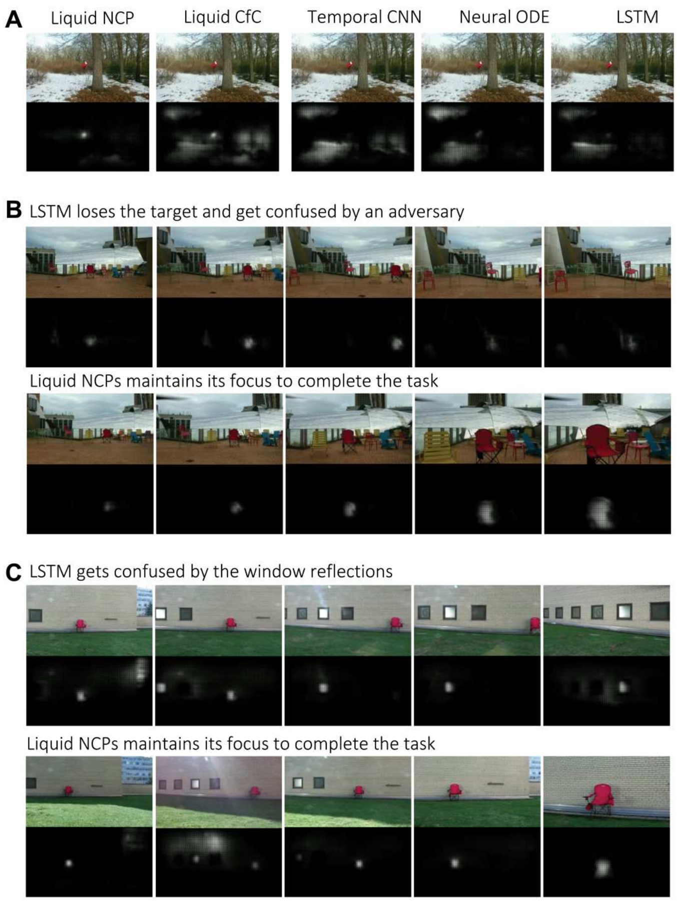
A
Liquid NCP
Liquid CfC
Temporal CNN
Neural ODE
LSTM
B LSTM loses the target and get confused by an adversary
Liquid NCPs maintains its focus to complete the task
C LSTM gets confused by the window reflections
Liquid NCPs maintains its focus to complete the task

Fig. 3. Evolution of flight-time networks' attention maps. (**A**) A frame illustration of the attention profile of different networks computed by the VisualBackProp saliency map computation algorithm (*44*). The top row shows the input image frame. The bottom row shows the saliency map. The brighter regions indicate a greater concentration of the convolutional layer's attention. Saliency scores range between 0 (black) and 1 (white). (**B**) A flight case study where an LSTM agent loses its target and gets confused by an adversary. Left to right shows the progress time. (**C**) A similar scenario as in (B), in a different OOD testing environment.

maps of the convolutional layers drastically improved for all networks when data augmentation was used during training.

The attention maps are greatly advantageous because they allow us to inspect and interpret reasons for a failed or successful decision-making process by a network under distribution shifts. For instance, we illustrate two OOD test cases in Fig. 3 (B and C), in which LSTM agents lost their target because of environmental adversaries, whereas, in very similar cases, liquid networks managed to keep their focus on the target and complete their tasks. We observed that learning more robust perception module filters (convolutional filters) can enhance the generalization of autonomous flight agents. This is why the robustness of the transformation from convolutional heads to the recurrent module (the decision-making compartment) highly depends on the choice of model. Our observations suggest that liquid networks improve representation learning on both of these fronts compared with other modern architectures.

Target's rotation and occlusion

We also conducted a series of experiments pertaining to the appearance of the target. More specifically, we evaluated the trained networks on the fly-to-target task in the Urban Lawn setting from the nominal distance of 10 m but first rotated and then partially occluded the target (with a black curtain, vertically along the middle). These experiments tested the robustness and invariance of the learned task representation that such alterations of the target can have on different architectures. The results for each of the networks are presented in Table S5.

We analyzed the results of these experiments relative to the third column of Table 1 to understand the effect of our modifications on performance. In the case of rotation, GRU-ODE is the architecture that suffered the most, managing only one success between both the 90° and 180° tests (2.5%), although it reached the upright target roughly a third of the time (32.5%) in the same environment. At both angles, LSTM saw its success rates halved compared with the upright scenario. Both NCP and ODE-RNN had their success rates reduced by around 40% on the 90° scenario, whereas CfC only lost 10%. In the 180° case, the latter three networks all maintained performance levels close to those in the initial upright setting. With the target occluded, only ODE-RNN and NCP managed to maintain their nominal performance (the latter even slightly improving on it). GRU-ODE, LSTM, and CfC saw performance reductions of 70, 60, and 30%, respectively.

Zero-shot hiking with adversaries

In this task, the target chairs were placed in the environment in the disposition depicted in fig. S9. The test was conducted under extremely bright sunlight, adding substantial perturbation to multiple network architectures' performance, with intense glare reflecting off of the hike targets. Moreover, we placed adversarial objects scattered in the scene (including a blue bin, a yellow chair, and two red chairs of different materials and builds). This setup required robustness in the face of extremely heavy distribution shifts in addition to proper task understanding to recognize and navigate toward the correct target at each step. The test was conducted on a cloudy day, with the target chairs' colors true to those appearing in the training data.

Table S6 contains the success rate results for this test. TCN and GRU-ODE models severely struggled to detect targets in this setup (less than 45% of runs reached the first checkpoint and none attended to the second). Liquid networks performed best in this scenario, with a CfC completing 14 of 20 runs of the hiking task with a target horizon of three. The results of this experiment show strong evidence that liquid networks have the ability to learn a robust representation of the task they are given and can generalize well to OOD scenarios where other models fail or perform poorly (see figs. S7 and S8 for examples of the perturbations and adversaries perceived by the drone's input camera). This observation is aligned with recent works (*21*) that showed that liquid networks are DCMs (*43*) and can learn robust representations for their perception modules to perform robust decision-making.

Triangular loop between objects

In this task, we demonstrated that our approach can generalize to an infinite number of hiking steps by setting up the three chairs as vertices of an equilateral triangle of sides equal to 10 m, as illustrated in Fig. 4. We programmed the quadrotor to turn an angle of 120° to the right after detection of a checkpoint (instead of flying up like in the experiments described above) to increase the likelihood of the next target appearing in the field of view. Using the CfC architecture, we

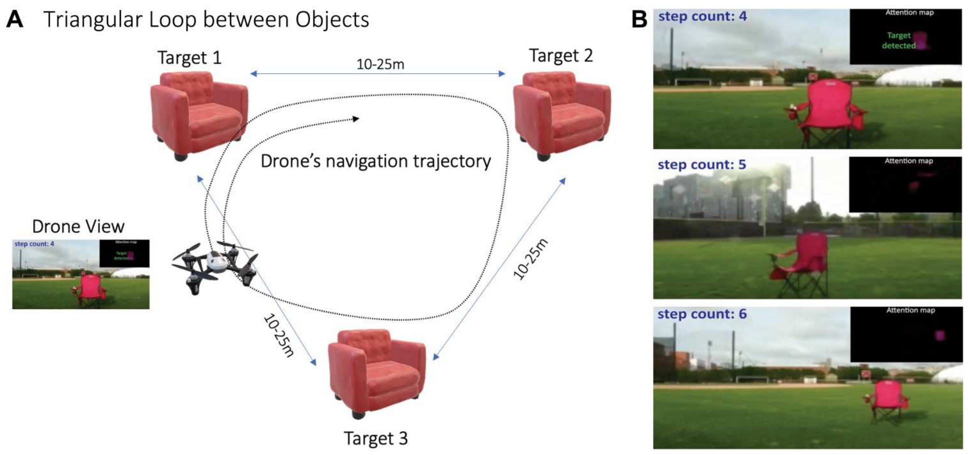

Fig. 4. Triangular loop between objects. (**A**) The triangular infinite loop task setup. (**B**) Time instances of drone views together with their attention map at steps 4, 5, and 6 of the infinite loop.

observed that the network could indefinitely navigate the quadrotor to new targets (we stopped after four laps, with a total of 12 checkpoints reached and detected).

Dynamic target

A useful task in quadrotor autonomous flight is following a moving target. We thus tested the trained policies' ability to pilot the drone in pursuit of a dynamic target. For this task, a figure of eight courses with six checkpoints was set out (at the extremities and either side of each loop, consecutive checkpoints being 5 m apart). This design choice ensured that run lengths were not upper-bounded, tested both turning directions, and included all possible natural lighting angles, all while containing the experiment in a constrained space. The target was moved from one checkpoint to the next, ensuring that enough reaction time in the camera field of view was granted, and the number of checkpoints reached was assigned as the score of a given test run. The experiment was reproduced in two environments, the Urban Lawn setting and a sports field we call Grass Pitch (see fig. S10). Testing in the latter environment is subject to highly challenging conditions of glaring sunlight and strong wind. In such conditions, most network architectures struggled to even latch onto the target (get to the first checkpoint and start following the target). We took this into account in our results by taking averages only on latched test runs but providing the rates at which each network initially detected the target.

Hence, table S7 shows that liquid networks consistently achieved longer trajectories in the Urban Lawn environment, ahead of LSTM and ODE-RNN architectures. In the Grass Pitch setting, all networks apart from LSTM, NCP, and CfC failed to latch onto the target in all attempts. In this challenging environment, liquid architectures marginally outperformed LSTM in terms of trajectory length, although CfC managed to latch onto the target more than twice as often as LSTM and NCP policies.

DISCUSSION

Choice of models matters for OOD generalization

Our experiments show large inequalities between different RNN architectures when performing a diverse range of fly-to-target control tasks in closed loop and all the more so when required to generalize in unseen and adversarial environments. Table 1 suggests that only five of the six networks were capable of reaching the target in the Training Woods environment on half or more of the attempts. TCN (7.5%) failed to achieve this threshold even on data from the training background distribution. These models also performed poorly in the other testing scenarios, with TCN unable to achieve a single success in any of the other three environments and GRU-ODE performing worse in comparison with the Training Woods scenario with the exception of Urban Lawn case, where it succeeded about a third of the time (32.5%). These two architectures exhibited both poor closed-loop performance and generalization and were thus deemed incapable of understanding and performing the control task assigned. Among the models performing reasonably well in both woods scenarios are the ODE-RNNs, with a 62.5% success rate when the target was in the same position as in the training data and up to 82.5% with the target placed in a slightly different position. Although this network seemed to have acquired the capability to achieve the task in closed loop, it generalized poorly to unseen environments, with 17.5 and 25% OOD success rates in the Urban Lawn and Urban Patio experiments.

Other models seemed to perform consistently in the woods, with success rates of 82.5% for LSTM, 100% for NCP, and 85% for CfC on the Training Woods test. In the Alternative Woods scenario, LSTM and both liquid networks (NCPs and CfCs) succeeded in reaching the target at a high success rate of more than 90%, showcasing the acquired ability to learn and execute closed-loop tasks from expert demonstrations.

When asked to generalize performance on the Urban Lawn, the top performer in this environment is CfC, which highly succeeded in attending to the target (90% success). On the task of flying to the target in the presence of natural adversaries and distractions in the extremely heavy distribution shift setup provided by the Urban Patio, LSTM succeeded in only 27.5% of the attempts, whereas both NCP (52.5%) and CfC (67.5%) achieved the best performance.

Furthermore, comprehensive evidence in favor of a crisp advantage for liquid neural networks for closed-loop end-to-end control learning was amassed through our extensive real-world experimental results. Our brain-inspired neural networks, and all the more so our CfC architecture, largely outperformed all other models on all of the range tests, the rotation and occlusion robustness tests, the adversarial hiking task, and the dynamic target tracking task.

When all nonliquid networks failed to achieve the task at twice the nominal distance seen in training, our CfC network achieved a 90% success rate and exhibited successful runs even at three times the nominal distance from the target (20%). Also, LSTM's performance was more than halved when the chair is rotated or occluded, whereas the CfC's success rate was only reduced by 10% for rotation (with success rates above 80%) and around 30% for occlusion (with success rates at 60%). Similarly, on the adversarial hiking task, CfC achieved a success rate more than twice that of its closest nonliquid competitor, completing 70% of runs, whereas the LSTM model only reached all three targets on 30% of occasions. Last, LSTM could track a moving target for an average of 5.8 steps, whereas our CfC network managed to follow for about 50% longer, averaging 8.8 checkpoints reached. Hence, our brain-inspired networks generalized to both new OOD environments and new tasks better than popular state-of-the-art recurrent neural networks, a crucial characteristic for implementation in real-world applications.

The greater robustness of liquid networks in the face of a data distribution shift was also confirmed through offline stress testing. We note that our brain-inspired networks were overall less prone to drifting from their nominal trajectories in the presence of artificially introduced image perturbations. Our architectures surpassed LSTM on noise, brightness, and contrast perturbations, with the CfC model outperforming all models on each of the perturbations introduced.

Understanding liquid network generalization to OOD settings

We associate the generalization performance of LTCs to two factors. During the end-to-end training, liquid networks impose an inductive bias on the convolutional filters of the head network, which enables a more robust feature representation associated with the task at hand. The saliency maps of the convolutional head networks are great empirical evidence of this fact. Elsewhere, liquid networks are DCMs at their cell level (*21*). The biological priors imposed on liquid networks form a dynamical system that enables mechanisms for regulating external and internal interventions, which are the necessary pillars of causal models (*36*, *45*, *46*). This way, during training, they manage to interact with the training data in a causal fashion to learn representations for both control and perception. In all tasks, we provide extensive evidence of their ability to drop irrelevant features under heavy distribution shifts and execute the task picked up on from training data, in this case, flying to the target regardless of the background profiles and other environmental variables.

Robustness in perception and control

The generalization of end-to-end imitation agents heavily relies on the data seen and the augmentation strategies in place to help encode the task in network weights. We observed that our data augmentation pipeline substantially improves the perception module of all networks and therefore enables them to obtain stable saliency maps with attention centered on the target (figs. S1 to S3). The augmentation, however, does not enable a robust execution of the control task upon change of test environments.

For instance, see the scenarios illustrated in Fig. 3 (B and C), in which the LSTM's perception module detects the target but still fails to navigate the drone. In these cases, the execution of the navigation task is hindered either by a distraction in the environment or by a discrepancy between the representations learned in the perception module and the recurrent module. This discrepancy might be behind the LSTM agent's tendency to fly away from the target or to choose an adversarial target instead of the true one detected by its perception module in the first place. With liquid networks, however, we noticed a stronger alignment of the representations between the recurrent and perception modules because the execution of the navigation task was more robust than that of other models.

Causal models in environments with unknown causal structure

The primary conceptual motivation of our work was not causality in the abstract; it was instead task understanding, that is, to evaluate whether a neural model understands the task given from high-dimensional unlabeled offline data. Traditional causal methods are based on known task structure or low-dimensional state spaces (for example, probabilistic graphical models) (*36*), whereas we can incorporate visual input into liquid networks. Therefore, causal modeling frameworks cannot be used in our high-dimensional tasks presented here, where the task structure is unknown. From the causality angle, liquid neural networks are DCMs: This was theoretically shown in Vorbach *et al.* (*21*).

Through extensive end-to-end closed-loop control testings, we showed a strong performance demarcation signal in favor of liquid neural networks against the advanced RNN models. This capacity to capture causality and contexts and learn reasoned representations for control originates from the incorporation of principled mechanisms for information propagation inspired by neural models of biological brains into our learning algorithms.

MATERIALS AND METHODS

Liquid networks: Brain-inspired neural models

A bottom-up approach to building brain-inspired neural models is to examine how neurons interact with each other through synapses in small biological brains (*19*, *20*). We can identify three mechanisms for information propagation that are abstracted away in the current building blocks of neural network-based controllers. First, neural dynamics are typically continuous processes described by differential equations (DEs) (*47*). Second, synaptic release is more than scalar weights: It involves a nonlinear transmission of neurotransmitters, the probability of activation of receptors, and the concentration of available neurotransmitters, among other nonlinearities (*48*). Last, the propagation of information between neurons is induced by feedback and memory apparatuses. These biological priors lead to the design of neural and synapse building blocks at a level of abstraction that is scalable and enriching with neural information processing attributes. The class of liquid neural networks LTC accommodates these three criteria (*35*).

LTCs (Fig. 5) are constructed by the interaction of leaky integrator neural models (*49*) with the steady-state dynamics of a conductance-based nonlinear synapse model (*47*). The model is a differentiable dynamical system with an input-dependent varying (for example, liquid) time characteristic. Their outputs are computed by numerical DE solvers when described by ordinary DEs and by continuous functions when described in closed-form

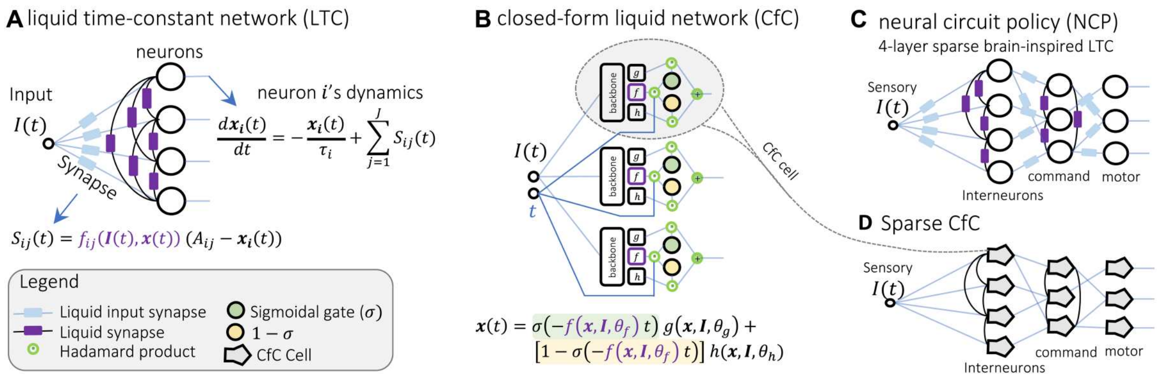
A liquid time-constant network (LTC)
neurons
Input
$I(t)$
Synapse
neuron i's dynamics
$\frac{dx_i(t)}{dt} = -\frac{x_i(t)}{\tau_i} + \sum_{j=1}^{J} S_{ij}(t)$
$S_{ij}(t) = f_{ij}(I(t), x(t))\,(A_{ij} - x_i(t))$
Legend
Liquid input synapse
Liquid synapse
Hadamard product
Sigmoidal gate (σ)
$1-\sigma$
CfC Cell
B closed-form liquid network (CfC)
backbone
g
f
h
$I(t)$
t
CfC cell
$x(t) = \sigma(-f(x, I, \theta_f)\,t)\,g(x, I, \theta_g) + [1-\sigma(-f(x, I, \theta_f)\,t)]\,h(x, I, \theta_h)$
C neural circuit policy (NCP)
4-layer sparse brain-inspired LTC
Sensory
$I(t)$
command
motor
Interneurons
D Sparse CfC
Sensory
$I(t)$
command
motor
Interneurons

Fig. 5. Liquid neural networks. (**A**) Schematic demonstration of a fully connected LTC layer (*35*). The dynamics of a single neuron *i* is given, where $\mathbf{x}_i(t)$ represents the state of the neuron *i* and τ represents the neuronal time constant. Synapses are given by a nonlinear function, where *f* (.) represents a sigmoidal nonlinearity and $A_i j$ is a reversal potential parameter for each synapse from neurons *j* to *i*. (**B**) Schematic representation of a closed-form liquid network (CfC) together with its state equation. This continuous-time representation consists of two sigmoidal gating mechanisms, σ and 1 − σ, with their activity being regulated by the nonlinear neural layer *f*. *f* acts as the LTC for CfCs. These gates control two other nonlinear layers, *g* and *h*, to create the state of the system. (**C**) A schematic representation of an NCP (*18*). The network architecture is inspired by the neural circuits of the nematode *C. elegans*. It is constructed by four sparsely connected LTC layers called sensory neurons, interneurons, command neurons, and interneurons. (**D**) CfC. A sparse neural circuit with the same architectural motifs as NCPs, this time with CfC neural dynamics.

approximations (Fig. 5B) (*50*). These brain-inspired models are instances of continuous-time (CT) neural networks (*35*, *41*) that can be trained via gradient descent in modern automatic differentiation frameworks. Liquid networks exhibit stable and bounded behavior, yield superior expressivity within the family of CT neural models (*35*, *41*), and give rise to improved performance on a wide range of time series prediction tasks compared with advanced, recurrent neural network models (*50*). In particular, a sparse network configuration composed of fewer than two dozen LTC neurons supplied with convolutional heads showed great promise in learning, end to end, to map high-dimensional visual input stream of pixels to robust control decisions (*18*). These liquid network instances are called NCPs, because their four-layered network structure is inspired by the neural circuits of the nervous system of the nematode *C. elegans* (*51*), as illustrated in Fig. 5 (C and D). In prior work, these sparse liquid networks learned how to navigate autonomous simulated aerial (*21*) and real ground (*18*) vehicles to their goals much more robustly than their advanced deep learning counterparts in a large series of behavioral-cloning experiments within their training distribution. The state-space representation of an LTC neural network is determined by the following set of ODEs (*35*):

$$\frac{d\mathbf{x}(t)}{dt} = -\left[\frac{1}{\tau} + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)\right] \odot \mathbf{x}(t) + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta) \odot A \tag{1}$$

Here, $\mathbf{x}^{(D\times 1)}(t)$ is the hidden state with size D, $\mathbf{I}^{(m\times 1)}(t)$ is an input signal, $\tau^{(D\times 1)}$ is the fixed internal time-constant vector, $A^{(D\times 1)}$ is a bias parameter, and ⊙ is the Hadamard product. Intuitively, LTC networks are able to change their equations on the basis of the input they observe. These networks, either in their ODE form or in their closed-form representation (*50*), demonstrate causality and generalizability in modeling spatiotemporal dynamics compared with their counterparts (*21*). Their closed-form representations are called closed-form continuous-time (CfC) models and are given by (*50*):

$$\mathbf{x}(t) = \underbrace{\sigma(-f(\mathbf{x}, \mathbf{I}; \theta_f)\mathbf{t})}_{\text{Time-continuous gating}} \odot g(\mathbf{x}, \mathbf{I}; \theta_g) + \underbrace{[1 - \sigma(-[f(\mathbf{x}, \mathbf{I}; \theta_f)]\mathbf{t})]}_{\text{Time-continuous gating}} \odot h(\mathbf{x}, \mathbf{I}; \theta_h) \tag{2}$$

Here, *f*, *g*, and *h* are three neural network heads with a shared backbone, parameterized by θ_f, θ_g, and θ_h, respectively. $\mathbf{I}(t)$ is an external input, and *t* is the time sampled by input time stamps, as illustrated in Fig. 5.

Throughout, liquid (neural) networks refer to the general category of models that are presented either by LTCs or CfCs. CfCs are closed-form liquid networks, and LTCs are ODE-based liquid networks.

Liquid networks capture causality

The key to liquid networks' robust performance under distribution shifts is their ability to dynamically capture the true cause and effect of their given task (*21*). This can be shown analytically because LTCs are DCMs (*21*, *43*), a framework through which models can account for internal and external interventions with independent mechanisms. Vorbach *et al.* (*21*) theoretically showed that the learning system described by an LTC network of Eq. 1 can control internal and external interventions by the network parameters θ and thus reduces to a DCM (*43*) as long as *f* is monotonically increasing, bounded, and Lipschitz continuous.

DCMs are probabilistic graphical models (*43*). DCMs differ from other causal models in that they do not cast around statistical dependencies from data directly; rather, they have dynamic mechanisms, such as the structure presented in Eq. 2, that enable extracting cause-and-effect from data (*52*).

Causal properties of DCMs focus the attention of LTCs on the task rather than the context of the task, and, for this reason, in this article, we hypothesize and show that tasks learned in one environment can be transferred to different environments for LTC networks where other models fail. More formally, consider a sequence of task steps **T**, a sequence of robot/world configurations

C, and a sequence of images **V**. Visual data are generated following a schematic graphical model:

$$\mathbf{T} \rightarrow \mathbf{C} \rightarrow \mathbf{V}$$

We can then explain how causal understanding implies knowing. Robot states cause images and not vice versa (so when training on images and control inputs, the network should output positive velocity when required for the task, not because it has learned that a sequence of images moving forward implies that the drone is moving forward). In addition, the drone's motion is governed by the task (for example, centering the chair in the frame) and not by other visual correlations in the data. With the data augmentation techniques implemented, we show that almost all networks can handle the former condition. The latter condition, however, is difficult to achieve for many networks, as extensively exhibited throughout this report. Our objective in designing fly-to-target tasks is to create a testbed where neither the concept of objects nor the task itself is given to the agent. The agents must learn/understand the task, the scene, and the object from offline data without any guidance on what the scene, object, and task are. In this setting, supervised learning of object detection, where we have to manually label bounding boxes, is out of scope because we aim explicitly not to use labeled data (bounding boxes). We show how to learn the task and scene end to end from data and generalize to unseen and heavily shifted environments.

To this end, we set out to empirically assess this property in a large series of in- and out-of-distribution end-to-end navigation tasks in challenging environments. We built advanced neural control agents for autonomous drone navigation tasks (fly-to-target). We explored their generalization capabilities in new environments with a drastic change of scenery, weather conditions, and other natural adversaries.

Training procedure

The following section details the preparation of the data and hyperparameters used for training the onboard models.

Data preparation

The training runs were originally collected as long sequences in which the drone moved between all five targets. Because sequences of searching for the next target and traveling between them could lead to ambiguity in the desired drone task, for each training run of five targets, we spliced the approach to each of the five targets into five separate training sequences and added these new sliced sequences to the training data.

Data augmentation

To increase the networks' robustness and generalization capabilities, we performed various image augmentations during training. The first set of augmentations were a random brightness shift with brightness offsets selected uniformly at random ranging from 0 to 0.4× of the max pixel brightness, a random contrast shift with a contrast factor between 0.6 and 1.4, and a random saturation multiplication between 0.6 and 1.4. Parameters for the aforementioned image augmentations were fixed within a sequence but randomized between sequences so that two images within the same training sequence had identical brightness, contrast, and

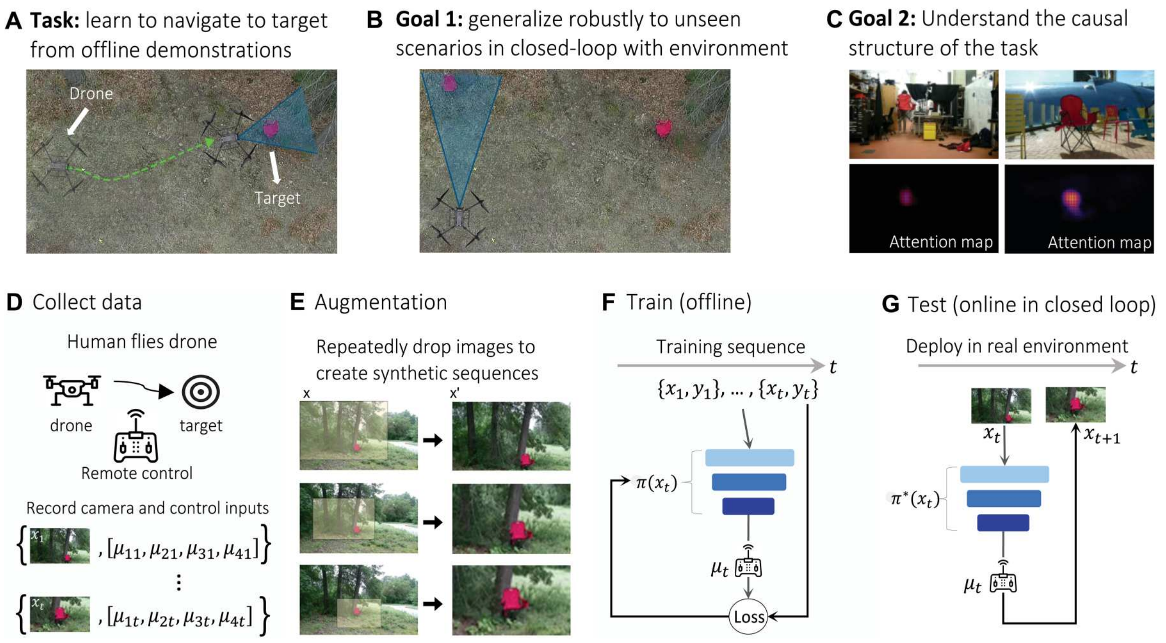
A Task: learn to navigate to target from offline demonstrations
Drone
Target
B Goal 1: generalize robustly to unseen scenarios in closed-loop with environment
C Goal 2: Understand the causal structure of the task
Attention map
Attention map
D Collect data
Human flies drone
drone
target
Remote control
Record camera and control inputs
$\{x_1, [\mu_{11}, \mu_{21}, \mu_{31}, \mu_{41}]\}$
$\{x_t, [\mu_{1t}, \mu_{2t}, \mu_{3t}, \mu_{4t}]\}$
E Augmentation
Repeatedly drop images to create synthetic sequences
x
x'
F Train (offline)
Training sequence
t
$\{x_1, y_1\}, \dots, \{x_t, y_t\}$
$\pi(x_t)$
μ_t
Loss
G Test (online in closed loop)
Deploy in real environment
t
x_t
x_{t+1}
$\pi^*(x_t)$
μ_t

Fig. 6. End-to-end learning setup. (**A**) Policies were trained to solve the following task: Using only images taken from an onboard camera, navigate the drone from its starting location to a target 10 m away, keeping the object in the frame throughout. (**B**) One goal of the work is to explore the generalization capability of various neural architectures by adapting the task to previously unseen environments. (**C**) Another goal is to understand the causal mechanisms underlying different networks' behaviors while completing the task, by visualizing networks' input saliency maps. (**D**) The training process started by hand-collecting human expert trajectories and recording camera observations and human expert actions. (**E**) To further convey the task, we then generated synthetic sequence by taking images collected during human flights and repeatedly cropping them, creating the appearance of a video sequence in which the drone flies up to the target and centers it in the frame. (**F**) The networks were trained offline on the collected expert sequences using a supervised behavior cloning MSE loss. (**G**) We then deployed trained networks on the drone in online testing and observed task performance under heavy distributional shifts, varying lighting, starting location, wind, background setting, and more.

saturation offsets, but two images in different sequences had different offsets. Last, each image had Gaussian random noise with a mean of 0 and an SD of 0.05 added.

In addition, to better convey the task, we performed closed-loop augmentation by generating synthetic image and control sequences and adding them to the training set. To generate these sequences, we took a single image with the target present and repeatedly cropped the image. Over the duration of the synthetic sequence, we moved the center of the crop from the edge of the image to the center of the target, causing the target to appear to move from the edge of the frame to the center. In addition, we shrank the size of the cropped window and upscaled all of the cropped images to the same size, causing the target to appear to grow bigger. Together, these image procedures simulated moving the drone to center the target while moving closer to the target to make it occupy more of the drone's view. We then inserted still frames of the central target with a commanded velocity of zero for 20 to 35% of the total sequence length. This pause sought to teach the drone to pause after centering the target. Algorithm S1 describes the synthetic sequence generation steps, whereas Fig. 6E visualizes the successive crops calculated by the algorithm and their resulting output frames.

To label synthetic sequences with velocity controls, we generated yaw commands equal to the horizontal distance in pixels between the target and the center of the frame times 0.01 and likewise generated throttle (up-down) commands equal to the vertical target offset times 0.01. We generated pitch (forward-backward) commands in the synthetic data proportional to the size of the cropped window. This technique increased the size of the training set by more than 2.5×, using a relatively small sample of images in which the target was present.

The initial position of the target within the frame was chosen randomly such that the *x* (horizontal) offset of the chair was between 10 and 70 pixels, and the *y* offset was between 5 and 40 pixels. (The size of the input images was 144 pixels by 256 pixels.] Synthetic sequences were later balanced such that the mean control signal across all generated data points was zero for the throttle and yaw channels. The sequence length was also randomized to be between 120 and 250 frames to encourage robustness to external forces or varying update frequencies. Note that although we used these augmented sequences for training, we did not include the closed-loop synthetic sequences in the validation set.

Hyperparameter tuning/network architectures

For each network architecture, we used tree-structured Parzen estimators (TPEs) (*53*, *54*) for tuning hyperparameters such as learning rate (LR), LR decay rate, and model-specific hyperparameters such as backbone unit count or connection seed that would maximize performance. The set of parameters leading to the lowest train + validation loss over the 40 episodes was selected for the final tested model.

TPE, a sequential model–based optimization algorithm, was responsible for sampling new hyperparameters to try across different episodes. After three trials, episodes with a higher median training + validation loss on the 10th epoch were pruned early and did not run to completion. Hyperparameter optimization was implemented using the Optuna library. Not all parameters were found via hyperparameter tuning; some parameters were hand-selected to shrink the dimensionality of the search space. The best hyperparameters found for each model architecture and used during testing are listed below in table S1.

For each chosen hyperparameter configuration, the number of trainable parameters in the corresponding model for every neural architecture is listed in table S3. Each network tested was prefixed by a simple CNN backbone for processing incoming images. The 128-dimensional CNN features were then fed to the recurrent unit that predicted control outputs. The shared CNN architecture is pictured in table S4.

Fine-tuning

To simultaneously learn the geospatial constructs present in the long, uncut sequences and the task-focused controls present in the cut and synthetic sequences, we fine-tuned a model trained on the long sequences with sequences featuring only one target. The starting checkpoint was trained on the original uncut sequences, all sliced sequences, and synthetic data corresponding to all targets. We then regenerated synthetic data with new randomized offsets for one target and fine-tuned on sliced sequences containing the one target and new synthetic augmented sequences that featured that target. Training parameters are listed in table S2.

Platform setup

We evaluated the proposed RNN architectures on a commercially available quadcopter with custom software running onboard. The system featured a forward-facing gimbal-mounted camera and a GPU to run onboard inference. We used Robot Operating System (ROS) (*55*) to handle communication between the RNN inference code and the lower-level drone flight controller.

Drone hardware

A DJI M300 RTK quadcopter was used for data collection and evaluation of policy performance. The M300 is an enterprise-grade drone that can be flown by manual control in its default configuration. It also interfaces with the DJI Manifold 2 companion computer, enabling programmatic control of the drone. The companion computer included an NVIDIA Jetson TX2, which has a CPU and GPU. The DJI Onboard SDK and its associated ROS wrapper provided an interface for giving the drone's low-level flight controller desired linear and angular velocity set points. The flight controller is a black box not modifiable by the end user, but it controls the four-rotor speeds to track the velocities specified by the TX2 computer.

A Zenmuse Z30 camera and gimbal were mounted on the underside of the drone, pointed forward. The gimbal compensated for the drone's current orientation such that the roll or pitch of the drone did not result in roll or pitch of the camera image. The gimbal followed the drone's current yaw such that the camera was always pointed forward. Images from the camera were available to the companion computer via the SDK.

System design

We implemented our software control system in ROS. A ROS node received images from the onboard camera, ran inference with the RNN controller to compute the new desired control on the basis of the current image, and sent the generated control outputs to the low-level flight controller using the onboard ROS SDK. The node has two threads: One is responsible for processing incoming images, and the other is responsible for running inference on the images and sending the resulting control signals to the flight controller. The TX2's GPU ran the RNN inference with GPU-accelerated Tensorflow 2.3.x. A depiction of the system design can be found in fig. S2.

Analysis methodology

VisualBackprop. A primary goal of this article was to analyze the task understanding of the policy networks. We visually analyzed the task understanding of a policy network by examining how the policy's CNN encodes and extracts features from an input. Each of the policies shares the same CNN head architecture, so differences in CNN weights between policies are representative of what the downstream architecture learns to extract from the environment.

Using the VisualBackProp algorithm, we used observations of the environment to compute attention mappings from the CNN head of the model architectures. The procedure, shown in Algorithm S4, is as follows: As input was passed through the CNN, each of the feature maps was averaged across the channels and saved. Starting from the final layer, the averaged feature map was deconvolved to the size of the previous layer's average feature map. The deconvolved feature map was multiplied by the previous layer's average feature map. This product was then passed to the previous layer. The deconvolution and multiplication were repeated iteratively, updating at each step until it reached the first layer. The result was a feature map displaying locations in the source image to which the model attends the most attention. These maps are the saliency maps (*56*).

Stress tests. For the stress tests in Fig. 2, we averaged the trajectory deviations for 10 runs recorded by the networks running in closed loop. Five runs were taken from the training environment, and five were taken from the urban test task. The perturbed runs were collected across a variety of different network types, including both liquid networks and conventional networks.

We only fed 40% of the frames in each run before measuring the final distance, because the longer the sequence, the more the trajectories diverge. Because the input images fed to the model were fixed, we could not model the change in perspective caused by the diverging viewpoints, and any estimates of future position became more inaccurate. To limit the effect of the offline frame generation while still maintaining enough frames to yield statistically significant results, we chose to only keep the first 40% of each sequence.

Attention-based hiking task setup. We designed a controller that relies on the attention maps perceived by the convolutional layers of each pretrained network architecture to trigger a hop between two targets. We observed that for networks with some understanding of the task when the target of interest is close enough, the saliency map displays a bright patch corresponding to the position of the object in the input image. We thus used an area occupancy criterion to trigger a switch of the target. While the network was piloting the quadrotor, we analyzed each saliency frame, masking pixels under a brightness threshold and detecting the most salient and largest connected patch. When the best candidate patch surpassed a fixed threshold in pixel area (corresponding to the drone being 2 m away from the camping chair), we considered the target reached and triggered an override of the network commands with a fixed-time throttle command to fly the quadrotor upward 1.6 m. This ensured that the current target left the field of view and that the network had access to images of the next segment of the scene. Control was given back to the network until a new target was detected and so on. This control protocol (depicted in fig. S3) can, by design, attend to an indefinite number of checkpoints, provided that the network sees and is capable of navigating to the next target.

Statistical analysis

The quantitative results of the paper presented in Tables 1 and 2 and table S4 to S7 all supply the number of active tests performed, *N*, and report the success rate of models in each scenario in percentages. On all compartments of Fig. 2, each point represents the mean across 10 runs, whereas the vertical bar on each point represents the SD corresponding to the data entry.

Supplementary Materials

This PDF file includes:
Materials and Methods
Figs. S1 to S10
Tables S1 to S7
References (*57*–*61*)

Other Supplementary Material for this manuscript includes the following:
Movies S1 to S4
MDAR Reproducibility Checklist

REFERENCES AND NOTES

1. Y. LeCun, Y. Bengio, Reflections from the turing award winners, in *International Conference on Learning Representations (ICLR)* (2020).
2. B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, N. Srebro, The role of over-parametrization in generalization of neural networks, in *7th International Conference on Learning Representations, ICLR 2019* (2019).
3. V. Vapnik, *The Nature of Statistical Learning Theory* (Springer science & business media, 1999).
4. S. Schaal, Is imitation learning the route to humanoid robots? *Trends Cogn. Sci.* **3**, 233–242 (1999).
5. S. Ross, G. Gordon, D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (JMLR Workshop and Conference Proceedings, 2011), pp. 627–635.
6. T. Zhang, Z. M. Carthy, O. Jow, D. Lee, X. Chen, K. Goldberg, P. Abbeel, Deep imitation learning for complex manipulation tasks from virtual reality teleoperation, in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 5628–5635.
7. E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, C. Finn, Bc-z: Zero-shot task generalization with robotic imitation learning, in *Conference on Robot Learning* (PMLR, 2022), pp. 991–1002.
8. P. Goyal, R. J. Mooney, S. Niekum, Zero-shot task adaptation using natural language. arXiv:2106.02972 [cs.AI] (5 June 2021).
9. F. Codevilla, M. Mü ller, A. López, V. Koltun, A. Dosovitskiy, End-to-end driving via conditional imitation learning, in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 4693–4700.
10. D. Ghosh, A. Gupta, A. Reddy, J. Fu, C. Devin, B. Eysenbach, S. Levine, Learning to reach goals via iterated supervised learning, in *International Conference on Learning Representations* (ICLR, 2020).
11. C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, P. Sermanet, Learning latent plans from play, in *Conference on Robot Learning* (PMLR, 2020), pp. 1113–1132.
12. S. Emmons, B. Eysenbach, I. Kostrikov, S. Levine, Rvs: What is essential for offline rl via supervised learning? arXiv:2112.10751 [cs.LG] (20 December 2021).
13. R. K. Srivastava, P. Shyam, F. Mutz, W. Jaśkowski, J. Schmidhuber, Training agents using upside-down reinforcement learning. arXiv:1912.02877 [cs.LG] (5 December 2019).
14. G. Neumann, J. Peters, Fitted q-iteration by advantage weighted regression, in *Advances in Neural Information Processing Systems* (Curran Associates Inc., 2008) vol. 21.
15. L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, I. Mordatch, Decision transformer: Reinforcement learning via sequence modeling, in *Advances in Neural Information Processing Systems* (2021), vol. 34.
16. S. James, M. Bloesch, A. J. Davison, Task-embedded control networks for few-shot imitation learning, in *Conference on Robot Learning* (PMLR, 2018), pp. 783–795.
17. C. Finn, T. Yu, T. Zhang, P. Abbeel, S. Levine, One-shot visual imitation learning via meta-learning, *Conference on Robot Learning* (PMLR, 2017), pp. 357–368.
18. M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, R. Grosu, Neural circuit policies enabling auditable autonomy. *Nat. Mach. Intell.* **2**, 642–652 (2020).

19. M. Lechner, R. Hasani, M. Zimmer, T. A. Henzinger, R. Grosu, Designing worm inspired neural networks for interpretable robotic control, in *2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019), pp. 87–94.
20. R. Hasani, M. Lechner, A. Amini, D. Rus, R. Grosu, A natural lottery ticket winner: Reinforcement learning with ordinary neural circuits, in *International Conference on Machine Learning* (PMLR, 2020), pp. 4082–4093.
21. C. Vorbach, R. Hasani, A. Amini, M. Lechner, D. Rus, Causal navigation by continuous-time neural networks, in *Advances in Neural Information Processing Systems* (2021), vol. 34.
22. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in *Advances in Neural Information Processing Systems* (NIPS, 2017), pp. 5998–6008.
23. S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, N. de Freitas, A generalist agent. arXiv:2205.06175 [cs.AI] (12 May 2022).
24. J. Kaplan, S. M. Candlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, D. Amodei, Scaling laws for neural language models. arXiv:2001.08361 [cs.LG] (23 January 2020).
25. S. Bubeck, M. Sellke, A universal law of robustness via isoperimetry, in *Advances in Neural Information Processing Systems* (NeurIPS, 2021), vol. 34.
26. J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, Diego de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, L. Sifre, Training compute-optimal large language models. arXiv:2203.15556 [cs.CL] (29 March 2022).
27. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. M. Candlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in *Advances in Neural Information Processing Systems* (NeurIPS, 2020), vol. 33, p. 1877.
28. V. Sanh, A. Webson, C. Raffel, S. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, A. Raja, M. Dey, M Saiful Bari, C. Xu, U. Thakker, S. S. Sharma, E. Szczechla, T. Kim, G. Chhablani, N. Nayak, D. Datta, J. Chang, Mike Tian-Jian Jiang, H. Wang, M. Manica, S. Shen, Z. X. Yong, H. Pandey, R. Bawden, T. Wang, T. Neeraj, J. Rozen, A. Sharma, A. Santilli, T. Fevry, J. A. Fries, R. Teehan, T. L. Scao, S. Biderman, L. Gao, T. Wolf, A. M. Rush, Multitask prompted training enables zero-shot task generalization, in *The Tenth International Conference on Learning Representations* (ICLR, 2022).
29. J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, Q. V. Le, Finetuned language models are zero-shot learners. arXiv:2109.01652 [cs.CL] (3 Sep 2021).
30. C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li, T. Duerig, Scaling up visual and vision-language representation learning with noisy text supervision, in *International Conference on Machine Learning* (PMLR, 2021), pp. 4904–4916.
31. J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, K. Simonyan, Flamingo: A visual language model for few-shot learning. arXiv:2204.14198 [cs.CV] (29 April 2022).
32. A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, M. Chen, Hierarchical text-conditional image generation with clip latents. arXiv:2204.06125 [cs.CV] (13 April 2022).
33. J. Zhang, K. Yang, C. Ma, S. Reiß, K. Peng, R. Stiefelhagen, Bending reality: Distortion-aware transformers for adapting to panoramic semantic segmentation, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2022), pp. 16917–16927.
34. A. Prakash, K. Chitta, A. Geiger, Multi-modal fusion transformer for end-to-end autonomous driving, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2021), pp. 7077–7087.
35. R. Hasani, M. Lechner, A. Amini, D. Rus, R. Grosu, Liquid time constant networks, in *Proceedings of the AAAI Conference on Artificial Intelligence* (AAAI, 2021), vol. 35, p. 7657.
36. B. Schölkopf, Causality for machine learning. arXiv:1911.10500 [cs.LG] (24 Nov 2019).
37. S. Hochreiter, J. Schmidhuber, Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).
38. Y. Rubanova, R. T. Chen, D. K. Duvenaud, Latent ordinary differential equations for irregularly-sampled time series, in *Advances in Neural Information Processing Systems* (NeurIPS, 2019), vol. 32.
39. K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches. arXiv:1409.1259 (3 Sep 2014).
40. C. Lea, R. Vidal, A. Reiter, G. D. Hager, Temporal convolutional networks: A unified approach to action segmentation, in *European Conference on Computer Vision* (Springer, 2016), pp. 47–54.
41. R. T. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (NeurIPS, 2018), pp. 6572–6583.
42. S. Bai, J. Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271 [cs.LG] (4 Mar 2018).
43. K. J. Friston, L. Harrison, W. Penny, Dynamic causal modelling. *Neuroimage* **19**, 1273–1302 (2003).
44. M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. J. Ackel, U. Muller, P. Yeres, K. Zieba, Visualbackprop: Efficient visualization of cnns for autonomous driving, in *IEEE International Conference on Robotics and Automation (ICRA)* (ICRA, 2018), pp. 1–8.
45. J. Pearl, Causal inference in statistics: An overview. *Stat. Surv.* **3**, 96–146 (2009).
46. W. Penny, Z. Ghahramani, K. Friston, Bilinear dynamical systems. *Phil. Trans. R. Soc. B Biol. Sci.* **360**, 983–993 (2005).
47. C. Koch, I. Segev, *Methods in Neuronal Modeling: From Ions to Networks* (MIT Press, 1998).
48. E. R. Kandel, J. H. Schwartz, T. M. Jessell, *Principles of Neural Science* (McGraw-Hill, 2000), vol. 4.
49. L. Lapique, Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *J. Physiol. Pathol.* **9**, 620 (1907).
50. R. Hasani, M. Lechner, A. Amini, L. Liebenwein, A. Ray, M. Tschaikowski, G. Teschl, D. Rus, Closed-form continuous-time neural models. arXiv:2106.13898 [cs.LG] (25 June 2021).
51. G. P. Sarma, C. W. Lee, T. Portegys, V. Ghayoomie, T. Jacobs, B. Alicea, M. Cantarelli, M. Currie, R. C. Gerkin, S. Gingell, P. Gleeson, R. Gordon, R. M. Hasani, G. Idili, S. Khayrulin, D. Lung, A. Palyanov, M. Watts, S. D. Larson, OpenWorm: Overview and recent advances in integrative biological simulation of *Caenorhabditis elegans*. *Philos. Trans. R. Soc. B* **373**, 20170382 (2018).
52. K. Friston, Causal modelling and brain connectivity in functional magnetic resonance imaging. *PLOS Biol.* **7**, e1000033 (2009).
53. J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in *Advances in Neural Information Processing Systems* (NIPS, 2011), vol. 24.
54. J. Bergstra, D. Yamins, D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in *International Conference on Machine Learning* (PMLR, 2013), pp. 115–123.
55. Stanford Artificial Intelligence Laboratory *et al.*, Robotic operating system.
56. M. Bojarski, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, U. Muller, K. Zieba, Visualbackprop: efficient visualization of CNNs. arXiv:1611.05418 [cs.CV] (16 November 2017).
57. S. Ross, D. Bagnell, Efficient reductions for imitation learning, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (JMLR Workshop and Conference Proceedings, 2010), pp. 661–668.
58. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction* (MIT press, 2018).
59. J. Chen, B. Yuan, M. Tomizuka, Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2019), pp. 2884–2890.
60. X. Li, P. Ye, J. Jin, F. Zhu, F.-Y. Wang, Data augmented deep behavioral cloning for urban traffic control operations under a parallel learning framework, in *IEEE Transactions on Intelligent Transportation Systems* (IEEE, 2021).
61. A. Amini, T.-H. Wang, I. Gilitschenski, W. Schwarting, Z. Liu, S. Han, S. Karaman, D. Rus, Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. arXiv:2111.12083 [cs.RO] (23 November 2021).

Acknowledgments: We would like to thank J. Feld, I. Futran, K. Mishra, P. Whartenby, C. Vorbach, T.-H. Wang, and W. Xiao for testing NCP and CfC models for confirmation of results across other domains. We also thank S. T. Monteiro of Aurora Flight Sciences for providing constructive feedback and validating our experimental setting throughout this project. **Funding:** This research was supported in part by the AI2050 program at Schmidt Futures (grant G22-63172), the United States Air Force Research Laboratory, and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein. This work was further partially supported by the Boeing Company. A.R. gratefully acknowledges support from a National Defense Science and Engineering Graduate Fellowship. **Author contributions:** R.H., A.R., P.K., M.C., and D.R. conceptualized, designed, performed research, and analyzed data. R.S., A.A., and M.L. contributed to designing research, data curation, research implementation, and analyzing data. D.R. helped with the design of the research and guided and supervised the work. All authors wrote the article. **Competing interests:** R.H. is also affiliated with the Vanguard Group. R.S. is currently affiliated with Altitude AI. P.K. is currently affiliated with Wing. D.R. is affiliated with the Mitre Corporation as a senior visiting fellow. This work in part has been filed as a provisional patent with application number 63/415,382 on 12 October 2022. The other authors declare that they have no competing interests. **Data and materials availability:** All data, code, and materials used in the analysis are openly available at https://zenodo.org/badge/latestdoi/610810400 and https://zenodo.org/badge/latestdoi/381393816 under Apache 2.0 License for purposes of reproducing and extending the analysis. The original hand-collected training dataset is published (http://knightridermit.myqnapcloud.com:8080/share.cgi?ssid=06lMJMN&fid=06lMJMN), with a file name of devens snowy and a fixed size of 33.2 GB. This dataset was used for training the starting checkpoints. The dataset used for fine-tuning the models tested on the drone can be found at http://knightridermit.myqnapcloud.com:8080/share.cgi?ssid=06lMJMN&fid=06lMJMN (devens chair, 2.3 GB) and has a subset of the full dataset containing only runs with the chair target. We have also included the exact synthetic datasets we used for our experiments. We have both a full dataset http://knightridermit.myqnapcloud.com:8080/share.cgi?ssid=06lMJMN&fid=06lMJMN (synthetic small4, 14.7 GB) used to train the starting checkpoint and a synthetic chair–only dataset http://knightridermit.myqnapcloud.com:8080/share.cgi?ssid=06lMJMN&fid=06lMJMN (synthetic chair, 4.3 GB) used to fine-tune the final models for online testing. We would like to thank IBM for providing access to the Satori computing cluster and MIT/MIT Lincoln Labs for providing access to the Supercloud computing cluster. Both clusters were very helpful for training models and hyperparameter optimization.

Submitted 6 May 2022
Accepted 22 March 2023
Published 19 April 2023
10.1126/scirobotics.adc8892

Robust flight navigation out of distribution with liquid neural networks

Makram Chahine, Ramin Hasani, Patrick Kao, Aaron Ray, Ryan Shubert, Mathias Lechner, Alexander Amini, and Daniela Rus

Sci. Robot., **8** (77), eadc8892.
DOI: 10.1126/scirobotics.adc8892